

CRIANDO UMA APLICAÇÃO JAVA WEB COM SERVLETS 2



CRIANDO UMA APLICAÇÃO JAVA WEB COM SERVLET

Programar aplicações Web para atualmente é uma aventura. Existe uma pilha de tecnologias e protocolos, conceitos e outros detalhes (HTTP, cookies, stateless, assincronicidade, etc) cujo entendimento é fundamental para a construções dessas aplicações.

Essa requisitos tecnológicos da aplicação são conhecidos normalmente como **requisitos não funcionais tecnológicos** ou somente **requisitos não funcionais**. São exemplos de requisitos não funcionais:

- **persistência em banco de dados;**
- **transação;**
- **web services;**
- **gerenciamento de threads;**
- **gerenciamento de conexões HTTP;**
- **cache de objetos;**
- **gerenciamento da sessão web;**
- **balanceamento de carga;**
- **entre outros.**

Além da questão tecnologica, aplicações atuais possuem regras de negócio bastante complicadas. Codificar essas regras já representa um grande trabalho.

É para contornar esses problemas que usamos o Java Enterprise Edition ou **Java EE**. O Java EE especifica uma série de componentes para o que o desenvolvedor possa focar apenas em implementar regras de negócio apoiado pelos recursos para gestão dos requisitos não funcionais da plataforma.

O principal componente dessa arquitetura é a **API de Servlets**. Essa API tem, de forma simplificada, o objetivo de receber chamadas HTTP, processá-las e devolver uma resposta ao cliente;

Cada **Servlet** é, portanto, um objeto que recebe requisições (**request**) e produz algo (**response**), como uma página HTML dinamicamente gerada, uma imagem ou apenas dados em formato .json. Um **Servlet** permite:

- Fazer com que uma classe seja acessível via navegador;
- Receber e converter parâmetros enviados;
- Distinguir os métodos HTTP;
- Executar suas lógicas e regras de negócio;
- etc.

A **Servlet** possui um ciclo de vida bem definido além de possuir métodos específicos associados a cada método HTTP (aqui, não podemos confundir métodos HTTP (post, get, put, delete), com os métodos de uma classe Java). Cada método da classe que representa a Servlet começa com o prefixo **“do”**: doGet(), doPost() e etc.

Cada método recebe como parâmetro, dois objetos. HttpServletRequest e HttpServletResponse.

O objeto **HttpServletRequest** é o responsável por representar uma requisição HTTP realizada por um cliente. O servidor recebe a requisição, cria um objeto dessa classe e chama o método adequado.

Dois métodos muito úteis dessa classe são:

- **String getParameter(String parameter)** → Obtém os parâmetros enviados por clientes por uma query string ou por formulário por exemplo;
- **String [] getParameterValues(String parameter)** → Obtém os parâmetros enviados por clientes por meio de formulário com campos multivalorados, por exemplo;

Se for implementado um algoritmo para resposta, o servidor obtém o objeto da classe **HttpServletResponse**, traduz para uma resposta HTTP e devolve ao cliente.

Outro recurso importante do Servlet são os recursos de “**delegação de encaminhamento**” e “**redirecionamento**”.

O **redirecionamento** é um tipo de resposta, em que o servlet pede para que o cliente cuide de redirecionar a página.

Já a **delegação de encaminhamento** ocorre completamente no lado do servidor e o resultado da ação de encaminhamento é transparente para o cliente.

RequestDispatcher	SendRedirect
Encaminha somente para outra servlet ou para um jsp (mesmo domínio);	Encaminha para um URL (outros domínios);
Permite anexar atributos na requisição;	Atributos ou dados são inseridos por query strings ;
Arquivo requisitado e o destino usarão a mesma requisição.	Arquivo requisitado e o destino NÃO usarão a mesma requisição.

veja mais detalhes [aqui](#).



Ainda temos o escopo de aplicações:

ESCOPO DE OBJETOS

∴ Servlets oferecem três níveis diferentes de persistência na memória:

- i. Contexto da aplicação:** vale enquanto aplicação estiver na memória
(`javax.servlet.ServletContext`)
- ii. Requisição:** dura uma requisição (`javax.servlet.ServletRequest`)
- iii. Sessão:** dura uma sessão do cliente (`javax.servlet.http.HttpSession`)

Assim para programaticamente temos:

```
//Obter o contexto da aplicação
ServletContext contexto = getServletContext();

//adicionar atributos ao contexto da aplicação
contexto.setAttribute("chave", "valor");

//obter atributos
String atributo = (String) contexto.getAttribute("chave");

//remover atributos
contexto.removeAttribute("chave");
```

Atividades

1. Implemente uma aplicação web que contenha um formulário para concorrer a uma vaga de emprego no Canadá. O formulário deve ter os seguintes campos:

- a)** Nome;
- b)** Data Nascimento;
- c)** Idioma nativo com as opções: inglês, espanhol e português (apenas um idioma pode ser escolhido - utilize um input do tipo select ou radio buttons);
- d)** Seleção de habilidades técnicas com as opções: Java, JavaScript, HTML e CSS (deve ser usado um checkbox - mais de uma opção pode ser marcada);

Programa uma Servlet que receba os dados desse formulário e mostre as informações devidamente formatadas para o usuário. Campos em branco ou não preenchidos não devem ser aceitos.

2. Implemente uma aplicação web que contenha os seguintes arquivos:

- a) index.jsp:** deve apresentar um formulário de login, com os campos usuário e senha;

b)LoginServlet

- i. Deve verificar se um login tem como o `usuario==admin` e `senha==admin`;
- ii. Se estiver correto, armazene **um atributo** no contexto da aplicação indicando que está logado;
- iii. Caso contrário, redirecione o usuário para o arquivo `index.jsp` com uma mensagem de erro;

c)OutraServlet e pagina.jsp

- i. Verificar se o usuário está logado;
- ii. Se estiver continua;
- iii. Caso contrário, redirecione ou dispatche o usuário para o arquivo `index.jsp` com uma mensagem de erro;

d)LogoutServlet

- i. Se houver atributos do usuário no contexto da aplicação, eles devem ser removidos.